Cours de C

Eric Berthomier Laurent Signac

Chapitre 1

Premiers pas

1.1 Prologue

Ce cours est destiné à vous faire connaître le C sous Linux.

1.2 Exemple de programme

Voici pour exemple un premier programme qui fonctionne malgré le fait qu'il ne soit pas normalisé. Celui-ci affiche le mot `Bonjour` à l'écran.

À l'aide de votre éditeur de texte favori, ouvrez un fichier nommé `programme1.c` (1^{e} programme du cours) puis tapez le texte suivant :

`programme1.c`

```
main ()
{
  puts ("Bonjour");
  getchar ();
}
```

À ce moment, si vous entendez hurler, crier, pleurer, vomir ou tout autre ignominie de ce genre, c'est normal, un puriste est passé derrière votre dos. Mais nous sommes là pour apprendre et il faut bien commencer par quelque chose.

Une fois le texte du programme frappé, il faut le compiler, c'est à dire en analyser la syntaxe.

Ceci se fait avec le compilateur `gcc`. Dans un premier temps, nous allons continuer dans le sale et compiler salement :

En ligne de commande, tapez : `gcc -c programme1.c`

Si vous n'avez pas fait d'erreur, ceci va vous donner un fichier nommé `programme1.o`. Ce fichier est nommé le fichier objet. Le fichier que vous avez passé en paramètre au compilateur se nomme le fichier source (source de tous les bonheurs de Linux ...).

Afin de pouvoir obtenir un fichier exécutable, il nous faut transformer le fichier objet en un exécutable. Pour cela, exécutez la commande suivante :

```
gcc -o programme1 programme1.o
```

Il ne reste plus qu'à exécuter le programme :

```
./programme1
```

Et comme par magie, s'affichera alors `Bonjour` et attendra que vous appuyez sur la touche Entrée.

Code source : le code source représente le programme sous sa forme textuelle (en langage C).

1.3 Normalisation du programme

Bon, je l'ai dit précédemment, nous avons fait dans le sale. Pourquoi me direz–vous ? Eh bien rendons notre compilateur bavard en lançant la commande `gcc -c -Wall programme1.c`. Observez la bornée d'insulte.

```
prog_1_1.c:5: warning: return-type defaults to ‘int’
prog_1_1.c: In function ‘main’:
prog_1_1.c:6: warning: implicit declaration of function ‘puts’
prog_1_1.c:9: warning: control reaches end of non-void function
```

Peu compréhensible et c'est normal.

En fait, l'option de compilation `-Wall` permet de « déclencher la production de messages soulignant toute technique autorisée mais discutable », en deux mots *à éviter*.

Nous allons donc normaliser ce programme.

À sa base, le langage C n'est qu'un ensemble de bibliothèques à partir desquelles le compilateur trouve les fonctions et les applications qui lui permettent de créer un programme exécutable. Exactement ce que vous faîtes lorsque vous recherchez dans une encyclopédie pour réaliser un exposé.

Certaines bibliothèque (les plus courantes) sont incluses dans des compilateurs ce qui permet à notre programme de se compiler. Normalement `puts` a besoin de la bibliothèque `stdio.h`. Pour ajouter une bibliothèque, il suffit d'ajouter `#include <nom de la bibliothèque>` en début de programme.

Le second point à corriger est l'absence de valeur de retour. La valeur de retour permet à un programme ou à l'utilisateur de savoir si le programme que l'on exécute s'est correctement terminé. En général 0 signifie une terminaison sans erreur.

En lui rajoutant quelques lignes on obtient donc :

programme2.c

```
#include <stdio.h>

int main ()
{
  puts ("Bonjour");
  getchar (); /* Permet d'attendre la frappe d'une touche */
  return (0);
}
```

La valeur de retour n'est pas obligatoire mais fortement conseillée. Pour ne pas utiliser de valeur de retour, on utilise `void main()` à la place de `int main()`. Le mot clé `void` peut se traduire par « ne contenant rien ».

Attention : Dans le cas de `void main()`, on utilise `return ;` et non `return(0) ;`.

Le programme devient donc :

programme3.c

```
#include <stdio.h>

void main ()
{
  puts ("Bonjour");
  getchar ();
  return;
}
```

La normalisation n'est pas finie, malgré tout. Pour être au plus pur, il faudrait écrire :

```
int main (int argc, char** argv)
```

Mais, pour être expliquée, cette formulation à elle seule nécessiterait tout le cours de programmation (les 15 cours ;-)). Donc nous en resterons là pour la normalisation.

1.4 Petit mot sur ce qu'est une bibliothèque

À l'instar de l'étudiant qui recherche dans des livres, on peut dire que le « .h » représente l'index du livre et le « .c » le contenu du chapitre concerné, le « .o » ou « .obj » n'étend que la forme pré compilée du « .c ».

Exemple : Lorsque le compilateur C rencontre le mot `puts`, il regarde dans chacun des « .h » déclaré par l'instruction `#include` si ce mot y est défini. Il trouve celui-ci dans la bibliothèque `stdio.h`. À l'inverse, s'il ne le trouve pas, celui-ci émettra une erreur de syntaxe.

1.5 Un exemple de fichier bibliothèque

Vous trouverez ci-dessous, un extrait de la bibliothèque `stdio.h`. On y retrouve notamment la déclaration de `puts` que l'on voit dans ce cours et la déclaration de `printf` que l'on verra dans le second cours. Vous trouverez ce fichier dans le répertoire `/usr/include`.

Extrait du fichier `stdio.h` :

```
/* Write formatted output to STREAM.  */
extern int fprintf __P ((FILE *__restrict __stream,
 __const char *__restrict __format, ...));
/* Write formatted output to stdout.  */
extern int printf __P ((__const char *__restrict __format, ...));
/* Write formatted output to S.  */
extern int sprintf __P ((char *__restrict __s,
 __const char *__restrict __format, ...));

/* Write formatted output to S from argument list ARG.  */
extern int vfprintf __P ((FILE *__restrict __s,
  __const char *__restrict __format,
  _G_va_list __arg));
/* Write formatted output to stdout from argument list ARG.  */
extern int vprintf __P ((__const char *__restrict __format,
 _G_va_list __arg));
/* Write formatted output to S from argument list ARG.  */
extern int vsprintf __P ((char *__restrict __s,
  __const char *__restrict __format,
  _G_va_list __arg));

/* Write a string, followed by a newline, to stdout.  */
extern int puts __P ((__const char *__s));
```

1.6 Les différentes fonctions

`puts` : permet d'afficher du texte suivi d'un retour à la ligne.
`getchar` : permet d'attendre la frappe d'une touche suivie d'un retour chariot ou un retour chariot seul.
`/* Commentaire*/` : permet de mettre un commentaire. On trouvera aussi `//` qui permet de mettre le reste de la ligne en commentaire.

Notre programme affiche bonjour et attend que l'on appuie sur une touche afin que l'on puisse voir ce qu'il a écrit.

1.7 Squelette de programme

On peut définir le squelette d'un programme C de la façon suivante :

```
/* Déclaration des bibliothèques */

int main ()
{
    /* Déclaration des variables */ cf. chapitre 2

    /* Corps du programme */
    getchar();/* Facultatif mais permet d'attendre l'appui d'une touche */
    return (0);/* Aucune erreur renvoyée */
}
```

1.8 Les blocs

La partie de programme située entre deux accolades est appelée un bloc. Je conseille de prendre l'habitude de faire une tabulation après le retour à la ligne qui suit l'accolade. Puis retirer cette tabulation après l'accolade fermante du bloc. Ainsi, on obtient :

```
{
    Tabulation
    Tout le code est frappé à cette hauteur
}

Retrait de la tabulation
Tout le texte est maintenant frappé à cette hauteur.
```

Cette méthode permet de contrôler la fermeture des accolades et de leurs correspondances.

1.9 Les commentaires

Commenter signifie qu'une personne ne connaissant pas le langage C doit pouvoir lire le programme et le comprendre. Les commentaires sont indispensables dans tout bon programme. Les commentaires peuvent être placés à n'importe quel endroit dans le programme. Ils commencent par `/*` et se terminent par `*/`.

```
/* Commentaire */
```

Vous trouverez aussi :

```
// Le reste de la ligne est un commentaire
```

1.10 Exercices d'application

1. Écrire un programme qui écrit au revoir.
2. Écrire un programme qui :

- Écrit « Salut toi, appuie sur une touche s'il te plaît »
- Attend l'appui d'une touche
- Écrit « Merci d'avoir appuyé sur une touche »

3. Commentez le précédent programme. Par exemple :
   ```
   puts ("Cours de programmation" ) ;
   /* Ecrit Cours de programmation à l'écran */
   ```
4. Écrire un programme qui écrit : «Hamlet says To be or not to be, that is the question.»

Corrigés des exercices du chapitre 1

Ecrire un programme qui écrit au revoir.

programme4.c

```
#include <stdio.h>

void main()
{
    puts("Au revoir "); /* Affiche Au revoir */
    getchar ();
}
```

Ecrire un programme qui :

1. Ecrit « Salut toi, appuie sur une touche s'il te plaît »
2. Attend l'appui d'une touche
3. Ecrit « Merci d'avoir appuyé sur une touche »

programme5.c

```
#include <stdio.h>

int main ()
{
    puts ("Salut toi, appuie sur une touche s'il te plaît");
    /* Affiche le message Salut toi, ...  s'il te plaît */

    getchar (); /* Attend la frappe d'une touche */

    puts ("Merci d'avoir appuyé sur une touche");
    /* Affiche le message Merci d'avoir appuyé sur une touche */

    return (0);
}
```

Commentez le précédent programme.
Déjà fait !

Ecrire un programme qui :
Ecrit : « Hamlet says To be or not to be, that is the question. »

programme6.c

```
#include <stdio.h>
#include <conio.h>
int main()
{
    puts ("Hamlet says To be or not to be, that is the question.");
    getchar();
    return (0);
}
```

Chapitre 2

Les variables (1^{e} partie)

2.1 printf : fonction indispensable pour afficher le contenu d'une variable

programme7.c

```
#include <stdio.h>
int main ()
{
    printf ("Coucou c'est moi\n");
    /* Affiche Coucou c'est moi à l'écran */
    getchar();
    /* Attendre l'appui d'une touche */
    return (0);
}
```

On pourrait dire que la fonction `printf` est la même que l'instruction `puts` vue précédemment mais il n'en est rien... Celle-ci est beaucoup, beaucoup, beaucoup plus puissante.

Syntaxe : La syntaxe de `printf` est très complexe et pourrait à elle seule faire l'objet d'un cours, nous en verrons donc des applications au fur et à mesure des besoins.

2.2 Variable

Comme son nom l'indique une variable est quelque chose qui varie. C'est vrai mais ce n'est pas suffisant. Une variable peut être considérée comme une boîte dans laquelle on met des données que l'on peut lire ou écrire.

La manière la plus facile de lire le contenu d'une variable est la fonction `printf` que l'on a aperçue précédemment. La manière la plus simple de donner une valeur à une variable est l'opérateur mathématique =. Écrire dans une variable ayant déjà une valeur revient à la modifier.

Une variable ne peut contenir qu'une seule chose à la fois. Si vous mettez une seconde donnée dans la variable, la précédente est effacée.

2.3 Déclaration d'une variable

La déclaration d'une variable se fait simplement en écrivant :

`<son type> <son nom>;`

Exemples de type de variables :
`char` : caractère
`int` : entier

2.4 Application : exemple

programme8.c

```
#include <stdio.h>
int main ()
{
    int i;
    /* i : variable de type entier */
    char  car;
    /* car: variable de type caractère */
    i = 65;
    /* i vaut 65 */
    car = 'E';
    /* car vaut E */
    printf ("i vaut %d.\n", i);
    /* Affiche la valeur de i */
    printf ("car vaut %c.\n",car);
    /* Affiche la valeur de car */
    getchar();
    /* Attendre l'appui d'une touche */
    return (0);
}
```

Explications :

1. On met dans la variable `i` la valeur 65.
2. On met dans la variable `car` la valeur de E.
   Note : En informatique, tout n'est que nombre, je dis donc la valeur de E et non E car c'est le code Ascii de E qui est sauvegardé dans cette variable. Nous reviendrons là dessus un peu plus tard.
3. `printf ("i vaut %d.\n", i);` : `%d` signifie que l'on attend une valeur entière, le programme va donc remplacer le `%d` par la valeur de `i`.
4. `printf ("car vaut %c \n", car);` : `%c` signifie que l'on attend une valeur de type caractère, le programme va donc remplacer le `%c` par la valeur de `car`. `\n` permet de réaliser un retour chariot c'est à dire un retour à la ligne.

2.5 Utilisation multiple du %

Le code « `%x` » signifie que le compilateur C doit remplacer ce code par la valeur correspondante (qui lui est fourni dans la suite de l'instruction) en la transformant dans le type `x`. Cette transformation est appelé un cast.

Exemple :

```
int i;
i =65;
printf ("Le caractère %d est %c",i,i);
```

nous donnera l'affichage suivant :
Le caractère 65 est `A`.
Le `%d` est remplacé par la valeur numérique de `i` c'est à dire 65.
Le `%c` est remplacé par la valeur alphanumérique (ASCII) de `i` c'est à dire `A` (cf. Table Ascii en Annexe).

2.6 Exercices d'applications directes

1. En utilisant ce qui a été fait précédemment, faîtes afficher les valeurs 70, 82, 185 et 30.
2. En utilisant ce qui a été fait précédemment, faîtes afficher, les caractères c, o, u, C, O, U.

2.7 Réutilisation d'une variable

On peut réutiliser une variable autant de fois que l'on veut, la précédente valeur étant effacée :
– `i = 3 ; i = 5 ; i = 7 ;` donnera au final pour valeur de `i` la valeur 7.
– `car = 'E' ; car = 'G' ; car = 'h' ;` donnera au final pour valeur de `car` la valeur de 'h'.

2.8 Caractères spéciaux

Certains caractères utilisés par la fonction `printf` (« `%` » par exemple) ou même tout simplement pour déclarer une variable (`'` pour les caractères par exemple) obligent à utiliser le caractère de suffixe \ pour pouvoir être affichés.

Exemple : Pour afficher un `%` avec `printf` j'écrirai :

```
printf("La réduction était de 20 \%") ;
```

Pour déclarer un caractère avec la valeur `'` (prononcée *quote* en informatique et non pas apostrophe (français)), on écrira :

```
char car ; car = '\'' ;
```

2.9 Exercices à réaliser

1. Faire un programme qui réalise l'affichage suivant en utilisant les variables.
   Aide : Sans retour chariot, on affiche à la suite :
   ```
   Coucou
   17
   ```
2. De la même façon, réaliser un programme qui réalise l'affichage suivant :
   ```
   C
   O
   U
   Cou
   1
   2
   3
   456
   C'est rigolo
   ```
   Rappel : Pour mettre une variable `c` égale à `'` on écrit `c='\''`
3. Ecrire un programme qui écrit :
   `« Hamlet says "To be or not to be, that is the question." »`
   avec les " bien sûr.

   Rappel : Pour pouvoir afficher un caractère de syntaxe C, par exemple ", on utilise le caractère \ comme préfixe à ce caractère. Pour obtenir un ", on utilise donc \".

Corrigés des exercices du chapitre 2

Faites afficher, en utilisant ce qui a été fait précédemment, les valeurs 70, 82, 185 et 30.

programme9.c

```
#include <stdio.h>

int main ()
{
    int i,a,b,c;
    i=70;
    a=82;
    b=185;
    c=30;

    printf ("i vaut %d.\n",i);
    printf ("a vaut %d.\n",a);
    printf ("b vaut %d.\n",b);
    printf ("c vaut %d.\n",c);
    getchar();

    return (0);
}
```

Faites afficher, en utilisant ce qui a été fait précédemment, les caractères c, o, u, C, O, U.

programme10.c

```
#include <stdio.h>

int main ()
{
    char a,b,c,d,e,f;
    a='c';
    b='o';
    c='u';
    d='C';
    e='O';
    f='U';

    printf ("a vaut %c.\n",a);
    printf ("b vaut %c.\n",b);
    printf ("c vaut %c.\n",c);
    printf ("d vaut %c.\n",d);
    printf ("e vaut %c.\n",e);
    printf ("f vaut %c.\n",f);
    printf ("a, b, c, d, e, f valent : %c, %c, %c, %c, %c, %c.\n",a,b,c,d,e,f);

    getchar();
    return (0);
}
```

Faire un programme qui réalise l'affichage :

```
Coucou
17
```

programme11.c

```
#include <stdio.h>

int main ()
{
    char car = 'C';
    int nbre = 17;

    printf ("%c",car);
    car = 'o';
    printf ("%c",car);
    car = 'u';
    printf ("%c",car);
    car = 'c';
    printf ("%c",car);
    car = 'o';
    printf ("%c",car);
    car = 'u';
    printf ("%c",car);

    printf ("\n%d\n",nbre);

    /* Attente */
    getchar();

    return (0);
}
```

Faire un programme qui réalise l'affichage :

```
C
O
U
Cou
1
2
3
456
C'est rigolo
```

programme12.c

```
#include <stdio.h>

int main ()
{
    char car = 'C';
    int nbre = 1;

    car = 'C';
    printf ("\n%c",car);
    car = 'O';
    printf ("\n%c",car);
    car = 'U';
    printf ("\n%c",car);
    car = 'C';
    printf ("\n%c",car);
    car = 'o';
    printf ("%c",car);
    car = 'u';
    printf ("%c",car);

    printf ("\n%d",nbre);
    nbre = 2;
    printf ("\n%d",nbre);
    nbre = 3;
    printf ("\n%d",nbre);
    nbre = 456;
    printf ("\n%d",nbre);
    car = 'C';
    printf ("\n%c",car);
    car = '\'';
    printf ("%c",car);
    car = 'e';
    printf ("%c",car);
    car = 's';
    printf ("%c",car);
    car = 't';
    printf ("%c",car);
    car = ' ';
    printf ("%c",car);
    car = 'r';
    printf ("%c",car);
    car = 'i';
    printf ("%c",car);
    car = 'g';
    printf ("%c",car);
    car = 'o';
    printf ("%c",car);
    car = 'l';
    printf ("%c",car);
    car = 'o';
    printf ("%c\n",car);
    /* Attente */
    getchar();

    return (0);
}
```

Faire un programme qui écrit :

« Hamlet says "To be or not to be, that is the question." »

programme13.c

```
#include <stdio.h>

int main ()
{

    printf ("Hamlet says : \"To be or no to be, that is the question.\"\n");

    /* Attente */
    getchar();
    return (0);
}
```

Chapitre 3

Les variables (2e partie)

Durant tout ce chapitre, nous aurons pour but simple, mais complet et complexe dans sa programmation, de faire une malheureuse calculatrice.

3.1 Exercice de mise en bouche

Ecrire un programme qui :
– écrit « Calculatrice : » et saute 2 lignes ;
– écrit « Valeur de a : » et saute 1 ligne ;
– attend l'appui d'une touche ;
– écrit « Valeur de b : » et saute 1 ligne ;
– attend l'appui d'une touche ;
– écrit « Valeur de a + b : » ;
Pas à pas, nous allons maintenant réaliser notre petit programme de calculatrice.

3.2 Déclaration des variables

Complétez le programme en :
– déclarant 2 variables `a` et `b` de type `int` (entier) ;
– assignant à ces deux variables les valeurs 36 et 54 ;
– faisant afficher le résultat de la somme de `a + b` (et non pas en faisant afficher 90 !).

Pour faire afficher le résultat il est possible d'utiliser la fonction `printf` directement ou indirectement en utilisant une troisième variable. Pour plus de facilité, nous allons utiliser un affichage direct. Celui-ci s'effectue de la façon suivante :

```
printf ("Valeur de a + b : %d",a+b) ;
```

Le `%d` est remplacé par la valeur de `a+b`.

3.3 Saisie des variables

Si une calculatrice se limitait à exécuter la somme de deux nombres fixes, le boulier serait encore de mise.

Pour saisir une variable, il est nécessaire d'utiliser la fonction `scanf`. La fonction `scanf` s'utilise de la façon suivante :
Saisie de la valeur `a` : `scanf ("%d", &a);`
Comme pour `printf`, on reconnaît le `%d` pour la saisie d'un nombre entier. Le `&` devant le `a` signifie que l'on va écrire dans la variable `a`.

Aïe... En fait `&a` signifie « l'adresse mémoire de `a` ». La fonction `scanf` va donc écrire dans l'emplacement mémoire (la petite boîte contenant la variable) de `a`. Si l'on oublie le `&`, on écrira chez quelqu'un d'autre, à une autre adresse. Et là ça peut faire mal, très mal...Mais ceci est une autre histoire.

Nous allons maintenant saisir les variables `a` et `b`. Pour exemple, je mets ici le code pour la saisie de `a`, la saisie de `b` reste à faire par vos soins à titre d'exercice.

```
/* Saisie de la valeur de a */
printf ("Valeur de a :\n");
scanf ("%d",&a);
```

Il est conseillé d'initialiser les variables avant de les utiliser. Au début du programme, après leur déclaration, assignez à `a` et à `b` la valeur 0.
Exemple : `a = 0;`
Si tout s'est bien passé, vous avez maintenant entre les mains une super calculatrice qui réalise des additions !

N.B. Appuyez sur la touche `Enter` (`Entrée`) après avoir tapé votre valeur.

Pour aller plus rapidement, il est possible d'initialiser une variable dans le même temps que sa déclaration. Pour cela, rien de plus simple, ajoutez à la fin de la déclaration `=` suivi de la valeur d'initialisation :
Exemple : `int i = 10;`

Evolution du logiciel : exercice Aïe aïe aïe, dur dur d'être informaticien. J'ai envie de faire une super calculatrice et je me dis qu'en fait la simplicité qui a eu lieue tout à l'heure n'est pas forcément adaptée à l'évolution désirée.

Déclarer une troisième valeur de type `int` (penser à l'initialiser à 0) que l'on nommera bêtement `s` comme somme. Une fois les valeurs de `a` et `b` saisies, initialisez `s` avec la valeur de `a + b`. Comme en mathématiques élémentaires. Affichez la valeur de `s`. On devrait avoir les même résultats qu'auparavant, bien sûr.

Exercices d'application : Réalisez 2 petits programmes qui font :
– la soustraction de 2 nombres ;
– la multiplication de 2 nombres.

Un nouveau type : les nombres à virgule ou flottants (`float`). Le type `float` permet de déclarer un nombre à virgule. Transformez les 3 précédents programmes en utilisant le type `float` au lieu du type `int` et `%f` pour saisir les valeurs de `a` et `b`, et pour afficher le résultat.

Petite aide : Ce petit morceau de programme saisit la valeur de `a` et la fait afficher :

```
float a;
printf ("Saisie de a :");
scanf ("%f",&a);
printf ("\na vaut : %f",a);
getch ();
```

Pour un affichage plus agréable il est possible de *fixer le nombre de chiffres après la virgule* de la façon suivante : `%.[nombre de chiffres après la virgule]f`
Exemple : `printf ("%.2f",a);`

Ouah, les 4 opérations !!! Créer un quatrième programme qui réalise la division de deux nombres. Tester avec une division par 0 !!! La solution à ce petit problème sera vu dans le cours sur les conditions (`if`).

Exercices à réaliser Compléter les 4 programmes afin de réaliser les opérations suivantes :

```
s = a + b + c
s = a -b-c
s = a / b / c
s = a * b * c
```

où `a`,`b` et `c` sont des nombres à virgules (`float`).

La fonction `abs` permet d'obtenir la valeur absolue d'un nombre entier. Utiliser cette fonction pour calculer la valeur absolue de (`a-b`).
Exemple d'utilisation de `abs` : `S = abs (a-b);`

La fonction `ceil` permet d'obtenir l'arrondi entier supérieur d'un nombre réel. Utiliser cette fonction pour calculer l'arrondi supérieur de (`a / b`).
Exemple d'utilisation de ceil : `S = ceil (a/b);`

Corrigés des exercices du chapitre 3

Soustraction de 2 nombres

programme14.c

```
#include <stdio.h>

int main ()
{
    int a,b;                          /* Déclaration des variables */
    int d;                            /* Différence entre les 2 nombres */

    a=0;                              /* Initialisation des variables */
    b=0;

    printf("Calculatrice :\n\n");
    printf("Valeur de a : ");
    scanf("%d",&a);
    printf("\n");
    printf("Valeur de b : ");
    scanf("%d",&b);

    d=a-b;
    printf("Valeur de a-b : %d\n",d);  /* Affichage de la différence */

    getchar ();
    return (0);
}
```

Multiplication de 2 nombres

programme15.c

```
#include <stdio.h>

int main ()
{
    int a,b;                        /* Déclaration des variables */
    int m;                          /* Résultat de la multiplication */
    a=0;                            /* Initialisation des variables */
    b=0;

    printf("Calculatrice :\n\n");
    printf("Valeur de a : ");
    scanf("%d",&a);
    printf("\n");
    printf("Valeur de b : ");
    scanf("%d",&b);

    m = a*b;
    printf("Valeur de a*b : %d\n", m);
    /* Affichage de la multiplication */

    getchar ();
    return (0);
}
```

Transformez les 3 précédents programmes avec le type `float` Aucune difficulté dans cet exercice, je ne mettrai ici la correction que de la multiplication, les autres opérations se réalisent sur le même schéma.

programme16.c

```
#include <stdio.h>

int main ()
{
    float a,b;                      /* Déclaration des variables */
    float m;                        /* Résultat de la multiplication */

    a=0;                            /* Initialisation des variables */
    b=0;

    printf("Calculatrice :\n\n");
    printf("Valeur de a : ");
    scanf("%f",&a);
    printf("\n");
    printf("Valeur de b : ");
    scanf("%f",&b);

    m = a*b;
    printf("Valeur de a*b : %f\n", m);
    /* Affichage de la multiplication */

    getchar ();
    return (0);
}
```

Pour l'addition :

```
m = a + b;
printf ("Valeur de a+b : %f", m);
```

Pour la soustraction :

```
m = a - b;
printf ("Valeur de a-b : %f", m);
```

Calculer la somme `a + b + c`

programme17.c

```
#include <stdio.h>

int main ()
{
    float a,b,c,s;                      /* Déclaration des variables */

    a=0;                                /* Initialisation des variables */
    b=0;
    c=0;
    s=0;

    printf("Saisie de a : ");
    scanf("%f",&a);
    printf("Saisie de b : ");          /* Saisie variables flottantes */
    scanf("%f",&b);
    printf("Saisie de c : ");
    scanf("%f",&c);

    s = a+b+c;                          /* Calcul de la somme */
    printf("Valeur de s : %.2f\n",s);  /* Affichage de la somme */

    getchar ();
    return (0);
}
```

Calculer la différence `a - b - c`

```
d = a-b-c;                              /* Calcul de la différence */
printf("Valeur de d : %.2f\n",d);   /* Affichage de la différence */
```

Calculer la multiplication `a * b * c`

```
m = a*b*c;                             /* Calcul de la multiplication */
printf("Valeur de m : %.2f\n",m);   /* Affiche la multiplication */
```

Calculer la division `a / b / c`

```
d = a/b/c;                              /* Calcul de la division */
printf("Valeur de d : %.2f\n",d);   /* Affichage de la division */
```

Calculer la valeur absolue de `a - b`

```
r=abs(a-b);                            /* Calcul de la valeur absolue */
printf("Valeur de r : %f\n",r);     /* Affiche la valeur absolue */
```

Calculer l'arrondi de `a + b`

```
c=ceil(a/b);                            /* Calcul de l'arrondi */
printf("Valeur de c : %f\n",c);    /* Affichage de l'arrondi */
```

Il aurait été possible d'utiliser `%d` du fait que l'arrondi est un nombre entier !

Chapitre 4

Les conditions

4.1 Exercice de mise en bouche

Ecrire un programme qui met en application le théorème de Pythagore pour calculer l'hypothénuse d'un triangle rectangle.

Rappel : Dans un triangle rectangle, l'hypothénuse (le plus grand côté) peut se calculer en appliquant la formule suivante :

$$\text{hypothénuse} = \sqrt{a^2 + b^2}$$

où a et b sont les longueurs des côtés adjacents à l'angle droit.

Notes : – La racine carrée s'obtient par l'utilisation de la fonction sqrt (valeur) contenue dans la bibliothèque math.h. (#include <math.h>)

– a^2 peut s'obtenir par a*a.

Méthodologie :

1. Rechercher les variables nécessaires et les déclarer dans le programme.
2. Faire saisir a au clavier.
3. Faire saisir b au clavier.
4. Effectuer l'opération de la racine carrée et afficher le résultat.

Attention : afin de pouvoir utiliser la bibliothèque mathématique du C (#include <math.h>), il est nécessaire d'ajouter au moment de l'édition de liens **-lm** ce qui nous donne : gcc -o monprog monprog.o -lm.

4.2 Les conditions : Si Alors Sinon

```
if (condition vraie)
{
    instructions 1
}
else
{
    instructions 2
}
```

```
si (condition vraie)
{    alors
    faire instructions 1
}
sinon
{
    faire instructions 2
}
```

Les conditions s'expriment avec des opérateurs logiques ...

4.3 Opérateurs logiques relationnels

Ils servent à comparer deux nombres entre eux.

Libellé	Opérateur
Inférieur	<
Supérieur	>
Equivalent	==
Différent	!=
Inférieur ou égal	<=
Supérieur ou égal	>=

4.4 Opérateurs logiques purs

Ce sont des opérateurs logiques permettant de combiner des expressions logiques.

Libellé	Opérateur
Et (and)	&&
Ou (or)	||
Non (not)	!

| `se nomme en informatique un pipe.`

4.5 Vrai ou faux

La valeur Vrai peut être assimilée à la valeur numérique 1 ou à toute valeur > 0. La valeur Faux peut être assimilée à la valeur numérique 0. L'opérateur Ou (||) correspond alors à une addition

Ou	Vrai	Faux
Vrai	Vrai	Vrai
Faux	Vrai	Faux

+	1	0
1	2	1
0	1	0

L'opérateur Et (&&) correspond alors à une multiplication

Et	Vrai	Faux
Vrai	Vrai	Faux
Faux	Faux	Faux

*	1	0
1	1	0
0	0	0

On notera que !Vrai = Faux et !Faux = Vrai.

4.6 Combinaison

Toutes les opérations logiques peuvent se combiner entre elles. La seule condition d'utilisation d'un si (if) avec de telles combinaisons est de l'entourer de ().

`Exemple :` if ((car == 'a') || (car == 'A'))

4.7 Astuce

Vous verrez souvent ce type de code écrit :

```
if (er)
{
    /* Alors faire quelque chose */
}
```

En appliquant ce qui a été vu précédemment, on en déduit que ce code signifie que

```
si (er != 0) /* si er différent de 0 */
{
    /* Alors faire quelque chose */
}
```

4.8 Les accolades

Les accolades entourant les blocs d'instructions d'une condition peuvent être omises si le bloc n'est constitué que d'une seule instruction.
`Exemple :`

```
if (car == 'b')
    printf ("car vaut b.");
else
    printf ("car est différent de b.");
```

4.9 Exercices

1. Faites saisir une variable de type entière et indiquez à l'utilisateur si celle-ci est positive ou négative ou nulle.

   ```
   Aide :
   if (a>0)
           printf ("Valeur positive");
   else
           printf ("Valeur négative");
   ```

2. Faites saisir une variable de type caractère et indiquez à l'utilisateur si celle-ci est une voyelle ou une consonne. On considèrera que le caractère saisi est en minuscule.

Corrections des exercices du chapitre 4

Ecrire un programme qui effectue le théorème de Pythagore

programme18.c

```
#include <stdio.h>
#include <math.h>

int main ()
{
    float a; /* base du triangle */
    float b; /* côté du triangle rectangle */
    float p; /* valeur de l'hypoténuse (p pour Pythagore !) */

    /* Initialisation des variables pour palier aux erreurs */
    a = 0;
    b = 0;

    /* Saisie de a */
    printf ("Valeur de la base : ");
    scanf ("%f",&a);

    /* Saisie de b */
    printf ("Valeur du côté : ");
    scanf ("%f",&b);

    /* Calcul par Pythagore */
    p = sqrt (a*a + b*b);

    /* Affichage du résultat */
    printf ("L'hypoténuse mesure : %.2f\n",p);

    /* Attendre avant de sortir */
    getchar ();

    return (0);
}
```

Test du signe d'une valeur saisie au clavier.

programme19.c

```
#include <stdio.h>

int main ()
{
    /* Valeur que l'on va saisir */
    int a = 0;

    /* Saisie de a */
    printf("Saisie de a : ");
    scanf("%d",&a);

    /* Test condition a<0 */
    if (a<0)
    {
        printf("la variable a est négative.\n");
    }
    else
    {
        /* Test condition a>0 */
        if (a>0)
        {
            printf("la variable a est positive\n");
        }
        /* Sinon a est nulle */
        else
        {
            printf("la variable a est nulle\n");
        }
    }

    getchar ();
    return (0);
}
```

Test si un caractère saisi au clavier est une consonne ou une voyelle.

programme20.c

```
#include <stdio.h>

int main ()
{
    /* Valeur que l'on va saisir */
    char car;

    /* Saisie du caractère a */
    printf("Saisie du caractère : ");
    scanf("%c",&car);

    /* Test condition car voyelle minuscule */
    if ((car == 'a') || (car == 'e') || (car == 'i') || (car == 'o') ||
    (car == 'u') || (car == 'y'))
    {
        printf("la variable car est une voyelle.\n");
    }
    else
    {
        printf("la variable car est une consonne.\n");
    }

    getchar ();
    return (0);
}
```

Chapitre 5

Mise au point

5.1 Prologue

L'objet de ce cours est de réaliser un petit break dans l'apprentissage du C et de s'attacher à voir que l'on est capable de réaliser avec le peu de moyen que l'on a.

Ce cours sera donc constitué de 3 exercices de difficultés croissantes avec apprentissage d'une nouvelle fonction et d'un exercice complet de programmation.

5.2 Exercice 1

Réaliser un programme qui saisisse un nombre et indique à l'utilisateur si celui-ci est plus grand ou plus petit qu'un autre nombre fixé par le programme.

`Exemple :`

```
si (nbre_saisi<10)
alors "plus petit"
```

Reprendre l'exercice du chapitre 4 qui disait si un nombre est positif, négatif ou nul.

5.3 Retour sur getchar()

La fonction getchar () permet d'attendre la frappe d'un caractère au clavier, de le lire et de le renvoyer. 2 utilisations peuvent être faites de getchar (), la première est celle permettant d'attendre la frappe d'une touche sans se soucier de sa valeur, la seconde est celle permettant de lire un caractère au clavier.

`Exemples :`

1. Attente
   getchar () ;

2. Saisie d'un caractère
   char car ;
   car = getchar () ;

A chaque fois, getchar () effectue le même traitement :
– Attend la frappe d'une touche au clavier suivi d'un retour chariot (Entrée).
– Renvoie le caractère frappé.
Dans le 1er cas, ce caractère n'est simplement pas récupéré.

5.4 Boucle Faire ... Tant que (vrai)

Do ... while, traduisez par Faire Tant que permet de réaliser une suite d'événements tant qu'une condition ou un ensemble de conditions est rempli.

`Exemple :`

programme21.c

```
#include <stdio.h>

int main ()
{
    char car;
    int sortie;

    do
    {
        printf ("Tapez S pour sortir !\n");

        /* On saisit un caractère */
        car = getchar ();

        /* On le compare pour savoir si l'on peut sortir */
        sortie = ((car == 's') || (car == 'S'));
    } while (!sortie);

    return (0);
}
```

`Rappel :`
– Un nombre entier vaut la valeur logique vraie si celui-ci est différent de 0.
– Un nombre entier vaut la valeur logique faux si celui-ci est égal à 0.
– || signifie un ou logique (or).

Remarque : Cette utilisation n'est pas très belle, le retour chariot utilisé pour la saisie du caractère étant renvoyé et interprété nous donne un affichage double. Malgré celà, au niveau de ce cours, nous nous en contenterons.

5.5 Exercice 2

Tapez l'exemple précédent, aménagez le, comprenez le, puis transformez le afin que l'on sorte de la boucle uniquement lorsque l'utilisateur a tapé le nombre 10.

Attention : La saisie d'un nombre ne se fait pas par getchar mais par scanf Cf. Chapitre 3.

5.6 Exercice 3

Voici un petit exemple de programme qui permet d'obtenir des nombres aléatoires entre 0 et 100.

programme22.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    int nb_alea; /* Nombre aléatoire */

    srand (time (NULL)); /* Initialisation des nombres aléatoires */

    /* Le nombre aléatoire est stocké dans une variable puis affiché */
    nb_alea = (int) (((float) rand () / RAND_MAX) * 100);
    printf ("%d",nb_alea);

    getchar ();
    return (0);
}
```

srand (time (NULL)) permet d'initialiser le système aléatoire.
rand () permet d'obtenir un nombre entre 0 et RAND_MAX.
(float) rand () permet de dire au langage C que l'on désire réaliser des calculs avec des nombres à virgules (flottants).
((float) rand () / RAND_MAX) permet d'obtenir un nombre entre 0 et 1, qui multiplié par 100 nous donne un nombre entre 0 et 100.

En vous aidant de ce petit programme et de ce qui a été fait précédemment, réaliser un petit jeu qui :

1. Initialise un nombre entre 0 et 100.
2. Tente de faire deviner ce nombre à l'utilisateur en lui indiquant s'il est plus petit ou plus grand.

Corrigés des exercices du chapitre 5

§5.2

programme23.c

```
#include <stdio.h>

int main ()
{
    int nb_choisi = 33;
    int nb_saisi = 0;

    printf ("Votre nombre : ");
    scanf ("%d",&nb_saisi);

    if (nb_choisi < nb_saisi)
        printf ("Mon nombre est plus petit.\n");
    else
    {
        if (nb_choisi == nb_saisi)
            printf ("Mon nombre est égal.\n");
        else
            printf ("Mon nombre est plus grand.\n");
    }

    /* Attente */
    getchar ();

    return (0);
}
```

§5.4

programme24.c

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int valeur;

    do
    {
        printf ("Votre nombre : ");
        scanf ("%d",&valeur);
    }
    while (valeur != 10);

    return (0);
}
```

§5.5

programme25.c

```
#include <stdio.h>
#include <stdlib.h>/* pour les valeurs aléatoires */
#include <time.h>

int main ()
{
    int nb_hasard = 0;
    int votre_nb = 0;

    srand (time (NULL));
    nb_hasard = (int) (((float) rand () / RAND_MAX) * 100); /* Nombre entre 0 et 100 */

    do
    {
        printf("Votre nombre : ");
        scanf("%d",&votre_nb);

        if (nb_hasard < votre_nb)
            printf ("\nMon nombre est plus petit\n");
        else
        {
            if (nb_hasard > votre_nb)

            /* il peut être aussi égal ... */
            printf ("\nVotre nombre est plus grand\n");
        }
    }
    while (votre_nb != nb_hasard);

    printf ("Trouvé\n");
    getchar ();

    return (0);
}
```